Pattern/Object Markup Language (POML): A Simple XML Schema for Object Oriented Code Description

Jason McC. Smith

Apr 7, 2004

Abstract

Pattern/Object Markup Language (or POML) is a simple XML Schema for describing object-oriented code in a unified manner with support for Design Patterns as well. (It can be used for pure procedural code also.) It supports class-based and object-based systems (and the two can be, and often are, mixed and matched.) Any OO (or procedural) language should be definable in POML, given enough thought. (This does not mean that every aspect of any language can be mapped to POML, only that the aspects that POML addresses should be representable in most any language.)

For instance, 'a = b + c' is best thought of as its compiler-representation equivalent: update(a, operator+(b, c)), and so on. C++, for example, is best translated to POML from a basic AST with class annotation. POML follows the basic principles of the sigma calculus, as described by Martin Abadi and Luca Cardelli in A Theory of Objects, Springer-Verlag, 1996, and the rho calculus as defined by the author in UNC-CS reports TR03-07, TR03-33 and other publications.

POML is a simple and yet expressive XML schema for describing object-oriented code in an open and language independent format for analysis and report generation. By using XML, POML allows simple XSLTs (eXtensible StyLesheet Transforms) to be written that can produce simple code examples in various programming languages, produce graphical or text reports, or offer data for source code analysis. It is for this latter task that POML was designed, as part of the System for Pattern Query and Recognition (SPQR). POML is the central format for shuttling information about source code among the various tools in SPQR, and provides a simple and easy format for manual double-checking.

The following XML schema is normative and validates against the W3C XML Schema Validator located at http://www.w3.org/2001/03/webdata/xsv, XSL version 2.7-1, dated Apr 1, 2004. A more convenient version may be found at http://www.cs.unc.edu/ smithja/spqr/POML.xsd.
Pattern/Object Markup Language (or POML) is a simple XML Schema for describing object-oriented code in a unified manner with support for Design Patterns as well. (It can be used for pure procedural code also.) It supports class-based and object-based systems (and the two can be, and often are, mixed and matched.) Any OO (or procedural) language should be definable in POML, given enough thought. (This does not mean that every aspect of any language can be mapped to POML, only that the aspects that POML addresses should be representable in most any language.)

For instance, ‘a = b + c’ is best thought of as its compiler-representation equivalent: update(a, operator+(b, c)), and so on. C++, for example, is best translated to POML from a basic AST with class annotation. POML follows the basic principles of the sigma calculus, as described by Martin Abadi and Luca Cardelli in _A Theory of Objects_, Springer-Verlag, 1996, and the rho calculus as defined by the author in UNC-CS reports TR03-07, TR03-33 and other publications. Copyright 2004, Jason McC. Smith, all rights reserved.
A name that has an optional 'scope' element. Since 'scope' is also of this type, nested scopes are supported transparently.

```xml
<xs:complexContent>
  <xs:extension base="xs:string">
    <xs:sequence>
      <xs:element name="scope" minOccurs="0" maxOccurs="1" type="scopeablename"/>
    </xs:sequence>
  </xs:extension>
</xs:complexContent>
```

A raw (instantiated) object. Useful for emulating class-based systems in a pure object notation. One can follow the example of Abadi and Cardelli in _A Theory of Objects_, and create an explicit object that contains the constructors, destructors, and static fields and methods for a class.

```xml
<xs:complexType name="object">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      A raw (instantiated) object. Useful for emulating class-based systems in a pure object notation. One can follow the example of Abadi and Cardelli in _A Theory of Objects_, and create an explicit object that contains the constructors, destructors, and static fields and methods for a class.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="nameditem">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="parent" type="type_reliance"/>
        <xs:element name="method" type="method"/>
        <xs:element minOccurs="0" maxOccurs="1" name="method" type="method"/>
        <!-- Implicit phi_reliance in field? -->
        <xs:element minOccurs="0" maxOccurs="0" name="uses" type="phi_reliance"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Descriptor for all instance-specific items in a class type definition. Contains inheritance information, methods, fields.

```xml
<xs:complexType name="class">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Descriptor for all instance-specific items in a class type definition. Contains inheritance information, methods, fields.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="nameditem">
      <xs:sequence maxOccurs="unbounded" minOccurs="0">
        <xs:element minOccurs="0" name="parent" type="type_reliance"/>
        <xs:element minOccurs="0" name="method" type="method"/>
        <xs:element minOccurs="0" name="field" type="field"/>
        <!-- Implicit phi_reliance in field? -->
        <xs:element minOccurs="0" name="uses" type="phi_reliance"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Methods have a name (nameditem), an optional list of parameters, an optional result, and possibly a 'static' tag indicating a class-level (as opposed to instance-level) ownership. A method can be tagged as 'abstract' (no definition), *or* it can include 'calls', 'uses' and 'update' relationships.

```xml
<xs:complexType name="method">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Methods have a name (nameditem), an optional list of parameters, an optional result, and possibly a 'static' tag indicating a class-level (as opposed to instance-level) ownership. A method can be tagged as 'abstract' (no definition), *or* it can include 'calls', 'uses' and 'update' relationships.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="nameditem">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="parameter" type="parameter"/>
        <xs:element name="result" type="resulttype"/>
        <xs:element minOccurs="0" maxOccurs="1" name="static"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```
<xs:complexType name="resulttype">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      A result is another named type that indicates whether it is passed back by reference (map).
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="nameditem">
      <xs:sequence>
        <xs:element name="passby" type="callbytype"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="update">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Update is the assignment operator. It has a left side (target), and a right side (source).
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="lhs" type="nameditem"/>
    <xs:element name="rhs" type="nameditem"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="field">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Fields are also named items, with an optional 'static' tag, and a required type, which is a scoped name.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="nameditem">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="static">
          <xs:complexType>
          </xs:complexType>
        </xs:element>
        <xs:element name="type" type="scopeablename"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="parameter">
  <xs:annotation>
  </xs:annotation>
</xs:complexType>
Parameters have a name, a type, and a 'keyword'. Some languages (such as Objective-C) have
required keyword support for arguments. In some (Python) it is optional, and in others (C++),
unknown. When translating an optional or no keyword language, simply make up a string unique
to the parameter within the method. 'kw1', 'kw2', 'kw3' and so on. This allows for mapping
external names to internal ones in a methodical manner.

A calling parameter is a parameter with one additional element, 'callby', mirroring the 'result'
element. It indicates whether a parameter is passed in via reference (map) or value (copy).

Corresponds to the 'calls' element of methods, and also the mu form reliance operator of SPQR.
Indicates a method calling another method, with a list of calling parameters.

The 'uses' element of methods, equivalent to the phi form reliance operator of SPQR.
Indicates that the holding method uses a field in some manner.
</xs:complexType>
</xs:complexType>
</xs:schema>